Patent
Docket No.: 57205US002

REUSABLE SOFTWARE COMPONENTS FOR INVOKING COMPUTATIONAL MODELS

TECHNICAL FIELD

The invention relates to computer modeling, such as computer modeling of manufacturing processes.

BACKGROUND

In any industrial manufacturing environment, accurate control of the manufacturing process can be essential. Ineffective process control can lead to products that fail to meet desired yield and quality levels. Furthermore, poor process control can significantly increase costs due to increased raw material usage, labor costs and the like. Accordingly, in an effort to gain improved control of the process, many manufacturers seek to develop computational models or simulations for the manufacturing process. A modeling expert, for example, may develop computational models using a variety of tools and a variety of modeling techniques including, for example, neural networks, linear regression, partial least squares (PLS), principal component analysis, and the like.

However, it is often difficult to effectively transfer the process knowledge produced by these models from the expert studying the model to the process engineers and operators on the manufacturing line. Reports produced by such models may be filed and forgotten, information from presentations and training sessions may not be retained, and process models may be left to languish on the computers of the experts who developed them. Furthermore, computational models are often developed using sophisticated modeling and simulation tools that are often unavailable at the manufacturing line, and are often too cumbersome and complex for use by the line operator.

SUMMARY

In general, the invention provides a system of reusable software modules that allow computational models to be seamlessly integrated directly into the manufacturing process. In particular, the software system encapsulates computational models into reusable objects. Reusable configuration and control modules can be embedded directly within conventional process management software for invoking and controlling the computational models in parallel. Accordingly, the system can be used to quickly and easily provide the results of one or more process models to line operators and process engineers during the manufacturing process, allowing the process engineer or line operator to more readily understand and adjust the manufacturing process.

In one embodiment, the invention is directed to a system in which a set of objects encapsulates computational models and provides generic interfaces for invoking the models. The system further comprises a software program executing within a computer operating environment and having an embedded control module to invoke the computational models in parallel. The system may further include a model aggregator to receive input values from the control module and to distribute the input values to the objects for use during model execution. The model aggregator may store configuration data that maps elements of an input vector to the various inputs of the models. Each element of the vector is referred to herein as an input slot. The configuration data may map an input slot of the model aggregator to multiple inputs of different models, thereby presenting a single interface by which the control module can provide data to, and receive data from, the objects.

In another embodiment, the invention is directed to a computer-readable medium containing instructions. The instructions cause a processor to instantiate a set of objects that encapsulate computational models and that include generic interfaces for invoking the computational models. The instructions further cause the processor to instantiate a model aggregator to distribute input values to the objects and to receive predicted output values from the objects. The instructions cause the processor to instantiate a control module to receive the output values from the model aggregator and to display the output values.

In another embodiment, the invention is directed to a method comprising encapsulating a set of computational models within objects, wherein each object provides a generic interface for invoking the encapsulated computational model. The method further comprises embedding a control module within an executable program, and invoking the set of objects from the control module to execute the computational models in parallel.

The invention may be capable of providing a number of advantages. By making use of the techniques described herein, computational models developed by a modeling expert, for example, can be encapsulated into reusable objects that a process engineer or other user can readily incorporate into an executable software program, such as a process management system within a manufacturing facility. In particular, the line operator, the programmer, and the process engineer need not have detailed knowledge of the models to make use of the models. In this manner, the line operator can execute the computational models directly and conveniently from the plant floor, using the process management software with which he or she is familiar.

The components of the system can be readily arranged as modular, reusable components to promote seamless integration with the executable program. The components may be implemented as, for example, ActiveX™ controls, Java applets, Java Beans™ and the like. Consequently, a user may embed the components within the process management software in a modular manner much in the same way other reusable software controls, such as buttons or text fields are often placed into a program and used.

The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Other features, objects, and advantages of the invention will be apparent from the description and drawings, and from the claims.

BRIEF DESCRIPTION OF DRAWINGS

FIG. 1 is a block diagram illustrating an example system in which process management software incorporates reusable software components for seamlessly invoking a plurality of computational models in parallel.

FIG. 2 is an object diagram illustrating exemplary classes and interfaces of the reusable software components.

FIG. 3 is a flow chart providing an overview of the operation of the system.

FIG. 4 is a flowchart illustrating a process of creating a set of objects to encapsulate computational models.

FIG. 5 is a flow chart illustrating operation of the system when invoking the computational models in parallel.

FIG. 6 illustrates an example user interface presented by a configuration module.

FIG. 7 illustrates another example user interface presented by the configuration module for setting specifications and targets of an output of a selected model.

FIG. 8 illustrates an example user interface displayed by a control module.

DETAILED DESCRIPTION

FIG. 1 is a block diagram illustrating an example system 2 in which process management software 4 incorporates reusable software components for seamlessly invoking a plurality of computational models. Although illustrated in reference to process management software, the reusable software components can be incorporated within any executable software program.

In general, process management software 4 executes within a computing environment provided by a computing device, such as a workstation or specific process control hardware. Process management software 4 provides signals 5 to manage manufacturing process 6 in order to produce product 7. Process management software 4 may, for example, provide parameters, thresholds, target levels and other information for use by manufacturing process 6. In addition, process management software 4 may monitor and record process data 9, which may include data indicative of a wide variety of process variables such as temperatures, processing times, speeds, thicknesses, flow rates, concentrations, weights and the like.

Configuration (config) module 8 and control module 10 are embedded within process management software 4 for seamlessly configuring and invoking a plurality of computational models 16 in parallel. In particular, configuration module 8 and control module 10 interact with model aggregator 12, which provides a single interface to a plurality of model objects 14A through 14N, collectively referred to as model objects 14. Each of model objects 14 "encapsulates" a respective computational model 16, and provides a generalized interface for interacting with the encapsulated model 16. In particular, model objects 14 encapsulate computation models 16 by including all of the information necessary to describe and execute computational models 16. Model objects 14 may contain, for example, the parameters, procedures, and algorithms necessary to interpret models 16. In this manner, a separate modeling environment or tool for executing models 16 is not needed. Model object 14 presents a generalized interface, allowing model aggregator 12 to communicate with the encapsulated models 16 in a common manner and format, even though models 16 may be developed using a wide variety of tools and modeling techniques.

Control module 10 invokes and provides input values to model objects 14 in parallel such that the encapsulated computational models 16 provide continuous predictive results. Consequently, model objects 14 and models 16 are not static, stand-alone models disconnected from manufacturing process 7. Rather, control module 10 makes use of the process data 9. In particular, control module 10 may continuously feed process data 9 into model aggregator 12, which distributes process data 9 to model objects 14 as appropriate. Similarly, model objects 14 continuously produce predicted output values as directed by control module 10. Control module 10 presents the predicted output values simultaneously to the line operator via display 18, e.g., in concert with measured data 9. In this manner, the line operator is able to observe predicted output values from a number of models 16 in parallel, along with the actual process data 9 used to feed the models 16. This feature enables the line operator to react more quickly to process events and control the manufacturing process in a more predictable manner.

Control module 10 also allows the line operator to adjust various inputs to the model objects 14 via input/output (I/O) device 19. In particular, as illustrated below, control module 10 may display a number of I/O controls, such as slide bars, toggle switches, or data entry fields by which the line operator can adjust the inputs to models 16. This allows control module 10 to display predicted output values of hypothetical process adjustments from multiple models 16 simultaneously, and in concert with process data 9 and user-provided data.

Configuration module 8 provides mechanisms by which a user, such as a process engineer, can easily select models 16 for encapsulation and parallel execution. In particular, configuration module 8 interacts with display 18 and I/O device 19 to allow the user to browse a computing environment for computational models, and select one or more of the models for encapsulation by model objects 14.

In addition, configuration module 8 allows the user to perform basic configurations for each model 16. Once a model is selected, for example, configuration module 8 displays the model inputs and outputs, and allows the user to set a minimum, maximum and target for each output. In other words, the user can provide configuration data that indicates when measurements from process data 9 fall outside of process specifications.

Furthermore, the user can define a vector for distributing input values to models 16. Each element of the input vector is referred to as an input slot. In particular, the user can define a set of input slots for passing input values to model aggregator 12, and can map the input slots to the various inputs of different models executing in parallel. In this manner, configuration module 8 allows the user to define a mapping between process variables and model inputs. Inputs for more than one model, for example, may be mapped to receive input from a common input slot, and the mapping may be arranged such that some of the input slots distribute input values to a subset of models 16. In other words, the user can develop an aggregate of input slots for distributing the input values to models 16 as appropriate. Model aggregator 12 stores the slot definitions, the mappings, as well as minimum, maximum and target values for each input, within configuration (config) data 15, shown in FIG. 1.

Prior to executing models 16, control module 10 communicates with model aggregator 12 to retrieve configuration data 15. For each output of models 16, control module 10 forms an appropriate output on display 18, such as a graph, that relates the output to the input slots. Control module 10 may determine, for example, the minimum and maximum of the X-axis of a graph based on the specification limits for the input as defined within configuration data 15. Control module 10 may display the outputs of models 16 in parallel by arranging the outputs in matrix form in which the outputs for each model form a row of graphs, and the defined input slots form the columns. Control module 10 may further indicate on each graph whether process data 9 falls within a specified range, as defined by configuration data 15.

Control module 10 and configuration module 8 are designed as modular, reusable software objects implemented according to a component architecture to promote seamless integration with process management software 4. Control module 10 and configuration module 8 may be implemented as, for example, ActiveX™ controls, Java applets, Java Beans™ and the like. Consequently, a user, such as a programmer, may embed control module 10 and configuration module 8 within process management software 4 in a manner much in the same way that other reusable software controls, such as buttons or text fields, are often placed into a program and used. Control module 10 and configuration module 8, however, are multi-featured tools for manipulating models and interacting with them.

FIG. 2 is an object diagram illustrating in further detail the classes and interfaces of model aggregator 12 and model objects 14 for seamless execution of a set of computational models in parallel. Model aggregator class 24 is responsible for instantiating existing sets of encapsulated models 16, as well as creating new model sets. The class provides a single point of control for the computational models. Discrete models class 26 maintains an enumerator for each of the models of the set. Discrete model class 28 provides an encapsulated model as well as an interface for invoking the model. In other words, model objects 14 of FIG. 1 represent instantiated objects of discrete model class 28.

Model aggregator 12 provides a persist interface 20 and a calculate interface 22 for communicating with configuration module 8 and control module 10, respectively. More specifically, configuration module 8 interacts with persist interface 20 for encapsulating, configuring and storing models. As an example, persist interface 20 may support the following methods:

LoadModelFile	Retrieve a collection of models from a file.
GetModels	Retrieve a reference to the set of models used for prediction.
AddModel	Add a model to the set of models.
SetInputNames	Set the names for the input slots of the set.
SetModelMap	Set a mapping between the slots to the models of the set.
GetModelMap	Retrieve the input variables specified by the models and arranged according to the currently defined mapping.
RemoveModels	Remove all discrete models from the set.
RemoveModel	Remove a single, specified model from the set.

Control module 8 interacts with calculate interface 22 for invoking and updating the contained discrete models 28. As an example, calculate interface 22 may support the following methods:

InputCount	Return the number of input variables for a model.
ModelCount	Return the number of models within the set.

GetModelType	Return the model type for the specified model in the set.
GetModelName	Return the model name for the specified model.
GetModelNames	Return the model names for the set.
GetInVariable	Return the stored configuration information for a given input variable.
GetOutVariable	Return the output variable name associated with a specified model.
Calculate	Invoke models using one or more sets of input variables. Return the output value(s).

The discrete model class 28 may support the following methods and public variables:

Name	A read/write variable indicating the model name.
Type	A string storing the type name of the model.
OutVariable	Returns the output variable name for the model.
InputCount	Returns the number of input variables associated with the model.
MinSpec	Retrieves and sets the minimum desired prediction value.
MaxSpec	Retrieves and sets the maximum desired prediction value.
GetInputs	Returns an array of input names for the model.
GetRange	Returns the minimum and maximum ranges for a given variable.

FIG. 3 is a flow chart illustrating the configuration and general operation of the software components of system 2. Initially, one or more users, such as modeling experts, develop computational models 16, possibly using a variety of modeling tools and techniques (40). During this process, the user typically selects a number of input variables that influence the process as well as one or more output variables to be predicted. The user then interacts with a modeling tool, such as MatLab™ by The MathWorks, Inc., of Natick, Massachusetts or other modeling tool, to develop a computational model that correlates the input variables and the output variables. The tool saves the model in a format that describes the inputs and outputs, as well as corresponding names and ranges. In this manner, the modeling tools may expose properties for the models to facilitate encapsulation, and in conformance with the requirements of modeling objects 14.

Alternatively, a translation program may be used to reformat the saved modeling information based on the requirements of modeling objects 14.

Next, a programmer embeds the configuration module 8 and the control module 10 within the process management software 4 (42) as described above. The user, such as the process engineer, then interacts with configuration module 8 via display 18 and I/O device 19 to select one or more computational models to be invoked and executed in parallel from the plant floor (44). For each selected model, configuration module 8 communicates with model aggregator 12 to create and store a corresponding model object 16 that encapsulates the functionality of the model and provides a generalized interface for interacting with the model (46). As described above, model objects 16 may comprise ActiveX™ controls, Java applets, Java Beans™ and the like. Finally, control module 10 communicates with model aggregator 12 to invoke the models in parallel, and to display the predicted results on display 18 simultaneously and integrated within a common user interface (48).

FIG. 4 is a flowchart illustrating in further detail the process of creating a set of objects 14 to encapsulate models 16. Initially, a user, such as a process engineer, interacts with configuration module 8 to select one or more computational models (50). Next, the user interacts with configuration module 8 to define one or more high-level input "slots" for the set of models (52). The user may define, for example, a number of different input slots corresponding to process data 9 that is measured from manufacturing process 6.

Next, configuration module 8 allows the user to map the various inputs for each model to the high-level slots (54). The user may, for example, map two or more inputs for the models 16 to receive the same process variable by associating the inputs with the same input slot.

The user may further interact with configuration module 8 to set specifications and targets for the outputs of the model (56). In particular, the user may define maximum and minimum values of a specification range for the output, as well as a target value for the output. Configuration module 8 communicates the configuration data to model aggregator 12 for creating and storing data for a corresponding model object 16 within configuration data 15 (58).

FIG. 5 is a flow chart illustrating the process of invoking computational models 16 in parallel. Upon execution of process management software 4, control module 10 instantiates model aggregator 12 (60) and directs model aggregator 12 to retrieve configuration data 15 defining a set of model objects 14 (61). Based on the retrieved data, model aggregator 12 instantiates a set of model objects 14 each object 14 according to the configuration data 15 (62).

After instantiating model objects 14, control module 10 communicates process data 9 and possibly user-supplied data to model aggregator 12 as input values (63,64), and instructs model aggregator 12 to invoke encapsulated models 16 and return predicted output values (65). Model aggregator 12 integrates the predicted output values from the various model objects 14 (66), and communicates the output to control module 10 (67). Control module 10 presents the output values received from model aggregator 12 on display 18, as well as actual process data 9 and any user-supplied input values (68). In this manner, control module 10 can display predicted output values from multiple computational models simultaneously, and in a manner that is seamlessly integrated with process management software executing on the plant floor.

FIG. 6 illustrates an example user interface 70 presented by configuration module 8 via display 18. In particular, user interface 70 includes a plurality of model selection fields 72 listing models currently identified by a user, such as a process engineer, as candidate models for encapsulation. In the illustrated example, the user has identified two models: (i) a thickness model implemented as a neural network, and (ii) a weight model implemented using Partial Least Squares (PLS) techniques.

To identify the models, the user can select the BROWSE button 74, causing configuration module 8 to display a dialog box from which the user can select one or more files containing one or more computational models. When the user presses the ADD button 69, configuration module 8 adds the identified model to workspace 71 for configuration. In this manner, the user can identify and selectively add models to the model set.

Configuration module 8 displays each of the selected models along a respective row of workspace 71. The columns of workspace 71 correspond to the defined input slots for model aggregator 12. Configuration module 8 allows the user to map the various inputs for each model to the input slots using reorder buttons 73A, 73B. In particular, the user can reorder the inputs for each model to move the inputs to an appropriate column, thereby mapping the inputs to the defined input slots. The user may, for example, map two or more inputs for the models to the same process variable by putting the inputs in the same column of workspace 71. Consequently, even if the inputs have different names within each model, the configuration module 8 can map the inputs to receive input values for the same process variable. Text boxes 75 allow the user to define labels for the slots for display to the line operator via control module 10.

The user can rearrange the rows of workspace 71 using MOVE UP button 77 and MOVE DOWN button 78. By rearranging the rows, the user can control the placement for the models within a user interface as displayed by control module 10. The user may remove models from workspace 71 by selecting REMOVE button 76.

To set specifications and targets for the outputs of the models, the user selects one of the models within workspace 71 and selects the SET SPECS button 79. FIG. 7 illustrates an example user interface 80 presented by configuration module 8 for setting the specifications and targets for the output of a selected model. In particular, user interface 80 includes data entry fields 82 and 83 for receiving maximum and minimum values of a specification range for the output, as well as data entry field 84 for receiving a target value for the output. Configuration module 8 communicates the data to model aggregator 12 for storage in configuration data 15 for later use by control module 10.

FIG. 8 illustrates an example user interface 90 produced by control module 10 on display 18. In particular, control module 10 communicates with model aggregator 12 to retrieve configuration data 15 including defined slot names, output variable names, and specified minimums, maximums and targets for the variables. In addition, user interface 90 displays actual process data 9 received from manufacturing process 6.

Specifically, control module 10 displays user interface 90 such that information for models 16 are arranged in matrix format. The outputs of each model 16 form the rows of user interface 90, with the defined input slots for model aggregator 12 forming the columns. As an example, user interface 90 displays outputs 92 from two models 16. In this manner, control module 10 displays outputs from multiple computation models 16 in parallel, and integrated into a common user interface.

For each model 16, user interface 90 displays predicted output values 95 as well as measured process values 96. Control module 10 formats the display of the predicted output values 95 based on the specifications set by configuration module 8. In particular, control module 10 may indicate that a predicted output value 95 has gone outside the specified limits by displaying the predicted output value as red, flashing or similar indicative format.

User interface 90 displays graphs 98 that illustrate the data correlating model inputs and outputs as received. In particular, model objects 14 communicate the data to control module 10 via model aggregator 12. User interface 90 indicates the current operating points, i.e., the points on the graphs that relate to the current input values and output values, by a set of intersecting vertical and horizontal lines.

For each input, user interface 90 provides adjustment mechanisms 100, such as sliders and data entry fields, by which the user can adjust the values provided to the input slots of model aggregator 12. As the user changes the inputs, control module 10 interacts with model aggregator 12 to instruct models 16 to recalculate graphs 98 and predicted output values 95. The user may change the limits for an output by entering the limits within the appropriate data entry fields of adjustment mechanisms 100.

Various embodiments of the invention have been described. These and other embodiments are within the scope of the following claims.